Cologne Chip

GateMate™ FPGA User Guide

Integrated Logic Analyzer

UG1005

🟩 User Guide August 2025

Cologne Chip AG
Eintrachtstr. 113
50668 Köln

Tel.: +49 (0) 221 / 91 24-0
Fax: +49 (0) 221 / 91 24-100

https://colognechip.com
info@colognechip.com

Contents

List of Figures

About this Document

This User Guide covers the getting started procedure of the integrated logic analyzer (ILA) for use with the Cologne Chip GateMate^{TM} FPGA Series and is part of the GateMate^{TM} documentation collection.

For more information please refer to the following documents:

- Technology Brief of GateMate^{TM} FPGA
- **DS1001** – GateMate^{TM} FPGA CCGM1A1 Datasheet
- **DS1002** – GateMate^{TM} FPGA Programmer Board Datasheet
- **DS1003** – GateMate^{TM} FPGA Evaluation Board Datasheet
- **UG1001** – GateMate^{TM} FPGA Primitives Library
- **UG1002** – GateMate^{TM} FPGA Toolchain Installation User Guide

Cologne Chip provides a comprehensive technical support. Please visit our website for more information or contact our support team.

1 Introduction

The integrated logic analyzer (ILA) can be used to perform in-system debugging of field-programmable gate array (FPGA) designs on the GateMate FPGA at runtime.

All signals of the design inside the FPGA can be monitored in a waveform.

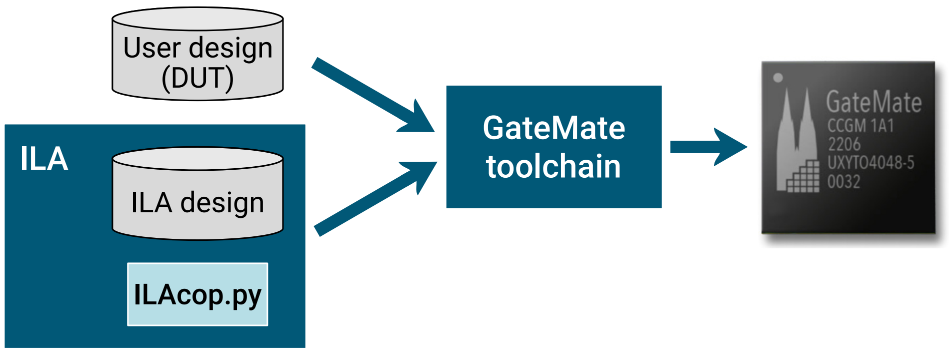

Figure 1: *Basic ILA usage*

The ILA can be used in conjunction with the GateMate FPGA and it consists of the ILA design and the ILA control program ILAcop.py.

The digital circuit of the ILA was designed in the hardware description language `Verilog`. A `Python` program controls the configuration of the ILA with the device under test (DUT) and the communication, as well as the interaction between the FPGA and the user during the debugging process.

`Python` 3 with additional packages is required for the execution of the user software. Further software, such as the GateMate FPGA toolchain and `GTKWave`, is also required to execute the ILA.

This project is licensed under the GNU General Public License. See the license file for details.

Please note!

It is assumed in this document

1. that the reader has developed an FPGA design and wants to use it on the GateMate FPGA and
2. that the use of GateMate FPGA, along with Yosys Open SYnthesis Suite (Yosys) and Place & Route, is well-known and the customer's design has already been loaded into the FPGA using the GateMate toolchain. Otherwise check the GateMate FPGA Toolchain Installation User Guide.

ILA can be used both with the GateMate^{TM} FPGA Evaluation Board and with the customers own hardware.

- If the GateMate^{TM} FPGA Evaluation Board is used, knowledge of its documentation is assumed. Please see **DS1003** – Evaluation Board Datasheet for more information.
- If customer hardware is used, please note that ILA requires a Universal Serial Bus (USB) interface between the computer and the GateMate FPGA. The FPGA ultimately uses an Serial Peripheral Interface (SPI) interface to communicate with the PCB circuitry. The USB-to-SPI adapter can either be built on the customer PCB or the GateMate^{TM} FPGA Programmer is used and connected to the SPI interface of the FPGA. Please see **DS1002** – GateMate Programmer Board Datasheet for more information.

2 Installation

2.1 Python 3 Installation

Python 3 is required to use the ILAcop.py. It is recommended to use at least Python 3.8 [1]. On most Linux systems, Python 3 can be installed from the package management software.

For example, users of Debian-based Linux distributions can use the *Advanced Package Tool*:

```
$ > sudo apt-get update
$ > sudo apt install python3
```

Windows users can also install Python 3 from the official Microsoft Store.

More information about the installation of Python 3 can be found at the Python download page.

2.2 Installation of ILA

2.2.1 ILA Download

Download ILA from the GitHub repository. The ILA root directory `gatemate_ila/` can be stored anywhere in the file system.

2.2.2 Repository Structure

In the following the files of ILA and their functions are explained:

`app/` This folder contains all the files required to run the ILA application and additionally all files created by the ILA software.

`example_dut/` This folder contains several examples which can be used to explore the use of ILA.

`log/` This folder contains the .log files generated by Yosys and Place & Route when bringing the gateware of both the ILA design and device under test (DUT) to the FPGA.

`net/` This folder contains the Verilog netlist of the gateware of the ILA design and DUT generated by Yosys.

`p_r_out/` This folder contains all files generated by Place & Route including the post-synthesis netlist of the gateware of both the ILA design and DUT.

`sim/` This folder contains the testbench, which can be used to simulate the gateware of the ILA design.

[1] ILA version 1.1.1 has been testetd with Python version 3.10.11.

`src/` This folder contains the `Verilog` source files of the ILA design.

`app/vcd_files/` This folder contains the .vcd files created by the ILA software. The .vcd files describe the waveforms of the analyzed signals and can be opened with a waveform viewer like `GTKWave`.

`LICENSE` The license file specifies the legal terms under which the contents of the repository can be used, modified, and distributed.

`app/ILAcop.py` This file is the main script from which the user starts the ILA software. The script handles the passed parameters and starts the processes requested by the user. For more information about the usage see Chapter 4.

`app/config.py` Here, configurations can be made for the used tools. For more informations see Chapter 5.

`app/requirements.txt` This file contains all the `Python` packages used by the ILA program. It can be used to install the packages using `Python`'s package manager `pip` (see Section 2.3).

`app/save_config/` All created configurations of ILA are stored in this folder in a JavaScript Object Notation (data interchange format) (JSON) file. ILA can be configured and started with the help of this files. The files contain all configurations in plain text and can be edited by the user. For more informations see Section 8.4.

`app/last_upload.txt` This file contains the path and filename to the JSON file of the last ILA configuration transferred to the FPGA. With this information, communication with ILA can be restarted using the last uploaded configuration.

`app/config_design/` During the configuration of ILA, the software creates an edited version of the DUT gateware. These `Verilog` files are stored in this folder.

2.3 Installation of External Python Packages

The ILA file `app/requirements.txt` contains a list of external `Python` packages that are required for the execution of the ILA software. These packages are:

- `pyftdi`
- `pyvcd`
- `prettytable`
- `pyusb`

One way to install these packages is to use the `Python` package management software `pip`.

If `pip` is not present, in some `Python` versions the module `ensurepip` is available, which can be used to install or upgrade `pip`:

```
$ > python3 -m ensurepip --upgrade
```

Or alternatively, pip can be installed via the operation system package management software. For Debian-based Linux distributions:

```
$ > sudo apt install python3-pip
```

Further information about the installation can be found on the Python installation page.

Finally, if pip is available, the required Python packages can be loaded. First, change to the gatemate_ila/app/ directory. From there, the file app/requirements.txt is used for package installation:

```
$ > pip3 install -r requirements.txt
```

2.4 Installation of a Wave Viewer

To view the waveform of the signals being analyzed, any wave viewer which is able to read .vcd files and that can be called in a console can be installed.

2.4.1 GTKWave

A very common wave viewer is GTKWave. On Debian-based Linux distributions the package management software can be used for installation:

```
$ > sudo apt install gtkwave
```

Windows users can download GTKWave in a .zip file. The files are extracted to any folder. This folder contains a folder bin/ with the file gtkwave.exe. Add the bin/ folder to the system path.

2.4.2 Other Wave Viewers

Other wave viewers can also be used if they fulfill the requirements mentioned above. The call of the corresponding program must be adapted accordingly in the app/config.py file: When customising the call, please note that the software places the .vcd file in the call between REPRESENTATION_SOFTWARE and REPRESENTATION_FLAGS.

2.5 Basic ILA Setup

The ILA program has to execute the toolchain applications and GTKWave. It must be able to call the execution files from the system path. Ensure that your system can recognize the following application calls from the system path by using the command line interface:

```
$ > yosys
$ > p_r
$ > openFPGALoader
$ > gtkwave
```

If a command is not found, its path must be entered in the system settings.

Alternatively, you can add the absolute path of the respective program to be executed in the `gatemate_ila/app/config.py` file. This ILA setup file contains some main settings.

For example, the file `config.py` could be changed as follows:

```
YOSYS        = '/home/dave/cc-toolchain-linux/bin/yosys/yosys'
YOSYS_FLAGS  = '-nomx8'

PR           = '/home/dave/cc-toolchain-linux/bin/p_r/p_r'
PR_FLAGS     = '-cCP +uCIO' # The +uCIO flag must not be removed.
                            # The ccf file is automatically appended

UPLOAD       = '/home/dave/cc-toolchain-linux/bin/openFPGALoader/
   ↪ openFPGALoader'
UPLOAD_FLAGS = ' -b gatemate_evb_spi -f --verify '

REPRESENTATION_SOFTWARE = '/home/dave/gtkwave/bin/gtkwave'
REPRESENTATION_FLAGS    = ['--save', 'save.gtkw']
```

More information about the `config.py` file is given in Chapter 5.

3 ILA Functionality

3.1 Trigger Condition

ILA has two ways of specifying a trigger condition:

Edge trigger: Any 1-Bit signal can be used as a trigger event on its falling or rising edge.

Pattern-matching trigger: Any set of 1-Bit signals and signal buses (or ranges thereof) that is part of the signal vector can be combined to form a trigger pattern. For each bit of this pattern, a specific state can be defined ('1', '0', or don't care). The trigger event occurs as soon as all bits have the defined value simultaneously.

The trigger condition is specified in the interactive shell before the DUT gets active. It can be changed, stopped or re-activated during runtime by the user.

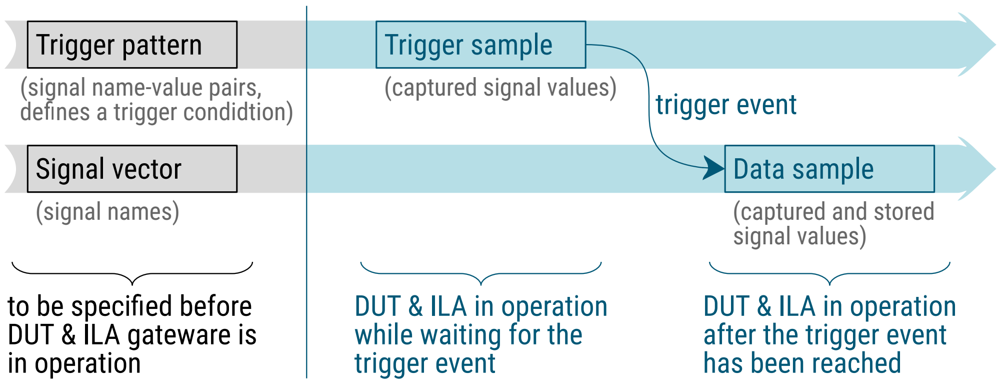
Trigger pattern
(signal name-value pairs, defines a trigger condidtion)
Trigger sample
(captured signal values)
trigger event
Signal vector
(signal names)
Data sample
(captured and stored signal values)
to be specified before DUT & ILA gateware is in operation
DUT & ILA in operation while waiting for the trigger event
DUT & ILA in operation after the trigger event has been reached

Figure 2: *Clarification of terms and contexts*

When the trigger condition has been reached, the data recording is executed until the specified number of data samples has been captured. Then the data is automatically downloaded to the computer and displayed in the wave viewer (see Figure 2).

Depending on the configuration, trigger monitoring is excecuted only once (see Figure 3), or several trigger events can be processed one after the other in a so-called trigger sequence (see Figure 4).

If the pattern-matching trigger is activated, more hardware is required for the ILA design and the maximum possible sampling frequency may be reduced.

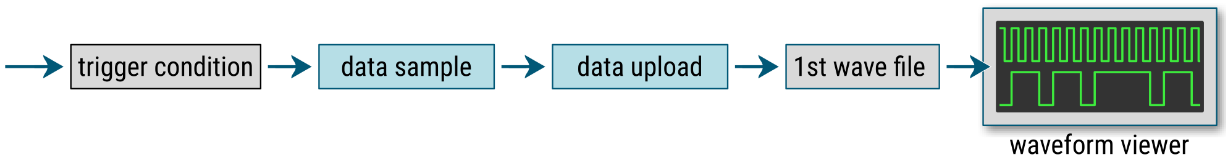

Figure 3: *Procedure with a single trigger event*

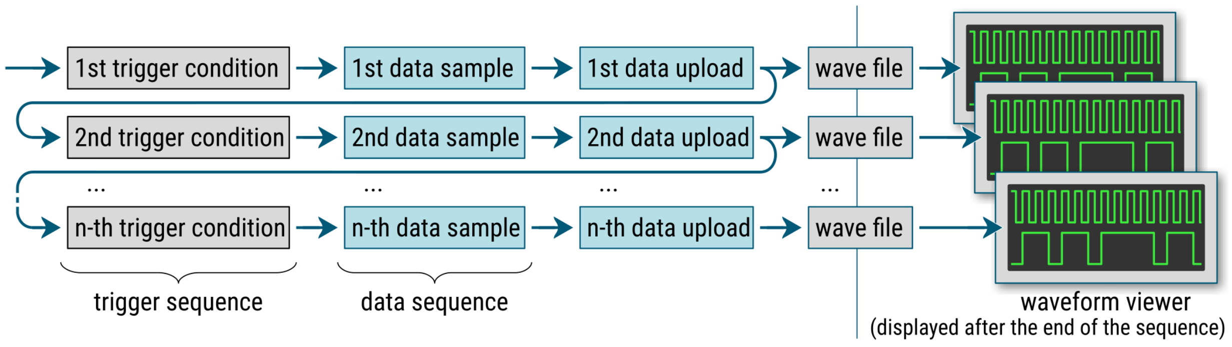

Figure 4: *Procedure with a trigger sequence*

If a trigger sequence is used, the time between the trigger events is measured and output. This generates the following console output after the complete data sequence have been recorded, for example:

```
### Times between trigger sequences ###
#                                      #
# start      - 1. trigger: 0.001346 s #
# 1. trigger - 2. trigger: 0.088927 s #
# 2. trigger - 3. trigger: 0.089089 s #
# 3. trigger - 4. trigger: 0.088873 s #
#                                      #
#######################################
```

3.2 Data Capturing

The signals to be captured are defined in the signal vector. For this purpose, the user can specify up to 1200 signal bits from the DUT design, which are stored in a signal vector for data capturing. The actual upper limit depends on the block RAMs used by the DUT.

These can be 1-bit signals or signal busses. It is also possible to select only partial sections of signal busses. All selected signals are combined in a signal vector.

To select a specific range of a signal bus, specify two natural numbers representing the bit positions. These numbers must lie within the valid range of the signal bus. The order in which these numbers are specified should match the definition of the bus. For example, if the bus is defined as `a[15:0]`, the range with the higher value is specified first, e.g., `10:3`. For a bus like `b[0:15]`, the order would be reversed, e.g., `3:10`.

Individual bits of a signal bus are simply indicated with the respective index. Several individual signals and ranges can also be specified in combination. For this, the indexes and ranges must be separated with a comma, such as

`12,10:8,4,2:0`.

There are three different ways to select signals for the ILA analysis:

1. Via the interactive shell:
All signals found in the DUT are listed in a table, including their bit width and the module name in which they were found. They can then be selected one after the other for analysis (see Section 8.3 from page 37).

2. Via the JSON file:
This method offers the advantage that signals can be conveniently entered using any editor. Further information can be found in Section 8.4 from page 52.

3. Directly in the design code via attribute marking:
Signals can be marked directly in the design code for analysis.

- In `Verilog`, it is sufficient to prefix the signal with an attribute, e.g.:

```
(* ILA *) reg [24:0] counter\_1;
```

- In VHDL, the attribute must first be declared, e.g.:

```
attribute ILA : boolean;
```

Individual signals can then be marked as follows:

```
signal ws2812\_out\_single : std\_ulogic;
attribute ILA of ws2812\_out\_single : signal is true;
```

ILA recognizes all signals marked in this way and automatically selects them for analysis. In the interactive shell, these signals already appear with an 'A' (for 'All') in the 'selected' column. The same applies to the automatically generated JSON file, in which the corresponding signals are also marked as 'A'.

3.3 Frequency Setup

The sampling frequency defines the delay between the stored data samples.

- The sampling frequency can be up to 160 MHz in low power and economy mode, and more than 200 MHz in speed mode.
- In general, the sampling frequency must be at least twice as high as the highest frequency of the signals to be sampled in order to reconstruct the signals correctly. If the DUT has a clock frequency that is twice or four times as high as the signals to be sampled, this is the perfect clock signal for ILA.
- The sampling frequency should be an integer multiple of the frequency of the signals to be sampled.
- However, synchronous signals can also be sampled at the same clock frequency at which they are generated by the DUT. This is necessary if the DUT operates at a very high clock frequency or if it does not have a clock frequency that is a multiple of the signals to be sampled. However, this requires that the signals to be sampled are synchronized with this clock frequency. This variant usually provides very reliable results, as there is no transition between different clock domains.

- For an unsynchronous DUT, the clock frequency of ILA should be at least four times as high as that of the DUT.

Please note: The higher the sampling frequency, the shorter the period of time to capture data in one piece, because the internal block RAMs fills up more quickly.

There are several ways to provide the ILA clock signal:

1. ILA can access the clock outputs of the FPGA-internal phase-locked loops (PLLs) used by the DUT. The user can select one of the up to 16 signals [2] and ILA establishes the connection to the Global Mesh for this signal via a `CC_BUFG` primitive. In principle, the GateMate FPGA can only route four signals in the Global Mesh. If more than four `CC_BUFG` nets are used in the overall design (which can also be done via the DUT alone), the Place & Route tool issues a warning and routes the surplus nets in the normal routing structure.
2. Alternatively, a GPIO input signal can be selected as the ILA clock.

   ILAcop.py automatically tries to find the clock sources in the top level of the DUT by using keywords. The first input signal that contains the keywords `clk` or `clock` in its name is automatically set as the clock signal. The user can change this default setting and select any other GPIO input instead. ILA connects the selected input signal with a `CC_BUFG` primitive to the Global Mesh.
3. Finally, if the DUT does not have a suitable clock, the ILA design can use its own PLL. This assumes that the DUT does not use all four PLLs of the GateMate FPGA. The PLL is configured via the interactive shell and ILA connects the PLL output signal with a `CC_BUFG` primitive to the Global Mesh.

The sampling frequency defines the delay between data samples. The ILA design has been successfully tested with sampling frequencies exceeding 200 MHz. The maximum sampling frequency can be increased by a smaller signal vector, as this may speed up the critical path.

If an additional PLL is set up, the following guidelines must be observed:

1. The desired ILA clock frequency can be entered in the interactive shell.
2. The default setting assumes that a frequency of 10 MHz is used as the external clock reference. If this is not the case, the reference frequency is specified with the `-f` option of ILAcop.py.
3. The PLL output offers four selectable phases in 90-degree steps. Any phase can be selected with the `-d` option of ILAcop.py (0 = 0°, 1 = 90°, 2 = 180°(default), 3 = 270°). Sampling always takes place on the rising edge.

3.4 Recording Length

During the ILA's recording process, the captured data is stored to the internal block RAMs of the GateMate FPGA. For this reason, the recording length depends on the number of freely available block RAMs.

[2] The GateMate FPGA has 4 PLLs with 4 clock outputs each.

It is $\mathrm{N_{RAM}} = 32$ the number of 40 k block RAMs of the GateMate FPGA CCGM1A1. Due to internal requirements of the ILA design, only a maximum of 30 block RAMs should be used, which is set in `config.py` by the default parameter setting '`available_BRAM = 30`' (see Section 5.8).

ILAcop.py determines the number of block RAMs occupied by the DUT and calculates the number of 40 k block RAMs available for ILA design with

$$\mathrm{N_{RAM\_ILA}} = \texttt{avaiable\_BRAM} - \mathrm{N_{RAM\_DUT}} \tag{1}$$

The maximum number of data samples $\mathrm{N_{smpl}}$ with small signal vectors where the number of bits in the signal vector (data width) is $w_{\mathrm{smpl}} \leqslant 20$, depends on[3]

$$\mathrm{N'} = \begin{cases} \mathrm{N_{RAM,ILA}} & , \ \mathrm{N_{RAM,ILA}} \leqslant 6 \\ 6 & , \ \mathrm{N_{RAM,ILA}} > 6 \end{cases} , \tag{2}$$

and it is

$$\mathrm{N_{smpl}} = \begin{cases} 32768 \cdot \mathrm{N'} & , \ w_{\mathrm{smpl}} = 1 \\ 16384 \cdot \mathrm{N'} & , \ w_{\mathrm{smpl}} = 2 \\ 8192 \ \cdot \mathrm{N'} & , \ 3 \leqslant w_{\mathrm{smpl}} \leqslant 5 \\ 4096 \ \cdot \mathrm{N'} & , \ 6 \leqslant w_{\mathrm{smpl}} \leqslant 10 \\ 2048 \ \cdot \mathrm{N'} & , \ 11 \leqslant w_{\mathrm{smpl}} \leqslant 20 \end{cases} . \tag{3}$$

For larger signal vectors with $w_{\mathrm{smpl}} > 20$, the maximum number of data samples depends on the limit factor

$$W = \left\lfloor \frac{\mathrm{N_{RAM,ILA}}}{\left\lceil \frac{w_{\mathrm{smpl}}}{40} \right\rceil} \right\rfloor \tag{4}$$

and it is

$$\mathrm{N_{smpl}} = 1024 \cdot \begin{cases} W & , \ W \leqslant 6 \\ 6 & , \ W > 6 \end{cases} . \tag{5}$$

ILA performs these calculations automatically and lets the user select the desired recording length from a list of possible durations, e.g.:

[3] The value 6 forms a decision limit that is based on the ILA's internal RAM structure. The block RAMs are cascaded in order to capture more data. The maximum cascading depth is 6 in order to meet the timing requirements. Therefore, the value 6 in equation 2 is to be understood as the upper limit for the depth of cascading.

```
!!!!!!!!!!!!!!!!!! Note !!!!!!!!!!!!!!!!!!!
!                                         !
! The capture duration must be defined.   !
! The maximum duration depends on:        !
!  – available ram                        !
!  – width of the sample                  !
!  – sampling frequency                   !
!                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Please choose one of the following durations:

-----Please choose one of the following durations: -----
+---+---------+---------------+--------------+------------+
| # | smp_cnt | duration [us] | FIFO Cascade |       FIFO |
+---+---------+---------------+--------------+------------+
| 1 |    1024 |         102.4 |        3 x 1 | 40 x 1024 |
| 2 |    2048 |         204.8 |        3 x 2 | 40 x 1024 |
| 3 |    3072 |         307.2 |        3 x 3 | 40 x 1024 |
| 4 |    4096 |         409.6 |        3 x 4 | 40 x 1024 |
| 5 |    5120 |         512.0 |        3 x 5 | 40 x 1024 |
| 6 |    6144 |         614.4 |        3 x 6 | 40 x 1024 |
+---+---------+---------------+--------------+------------+

Total Capture duration (choose between 1 and 2): 2

############# Capture duration ##############
#                                          #
# Sample count = 8192                      #
# Capture duration = 819.2 us              #
#                                          #
############################################
```

3.5 Input Control

One input signal or signal bus of the top-level entity of the DUT can be overwritten. Please note that this signal is then no longer connected to the IO pin(s) of the FPGA. If this function is activated, a binary value (1 or 0) can be assigned to the relevant signal, which is used as a constant value at runtime. This is set before the start of data capturing.

If a signal bus is selected, the whole bus or only partial sections can be overwritten with all 0's or all 1's. The bit selection is done in the same way as described in Section 3.2.

However, only one input signal or signal bus can be selected for this function.

3.6 Reset Control

Normally, ILA has no influence on the reset control of the DUT. ILA and DUT are started automatically after gateware upload. ILA uses the CC_USR_RSTN primitive for reset. The DUT is started via the reset condition implemented in the DUT. This can also be the

CC_USR_RSTN primitive or any dedicated I/O input signal of the FPGA as shown in Figure 5. Since ILA needs some time after startup to begin data capturing, the DUT cannot be observed during this startup phase in normal operation mode.

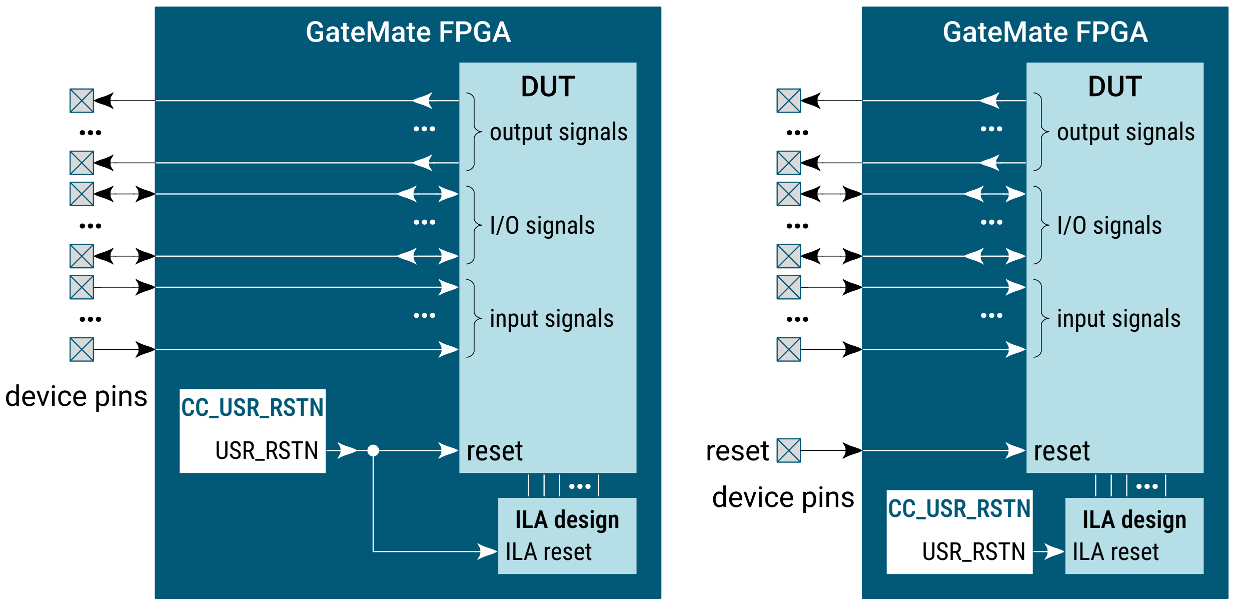

Figure 5: *DUT reset in normal operation mode (left: reset together with ILA via primitive, right: reset via input pin)*

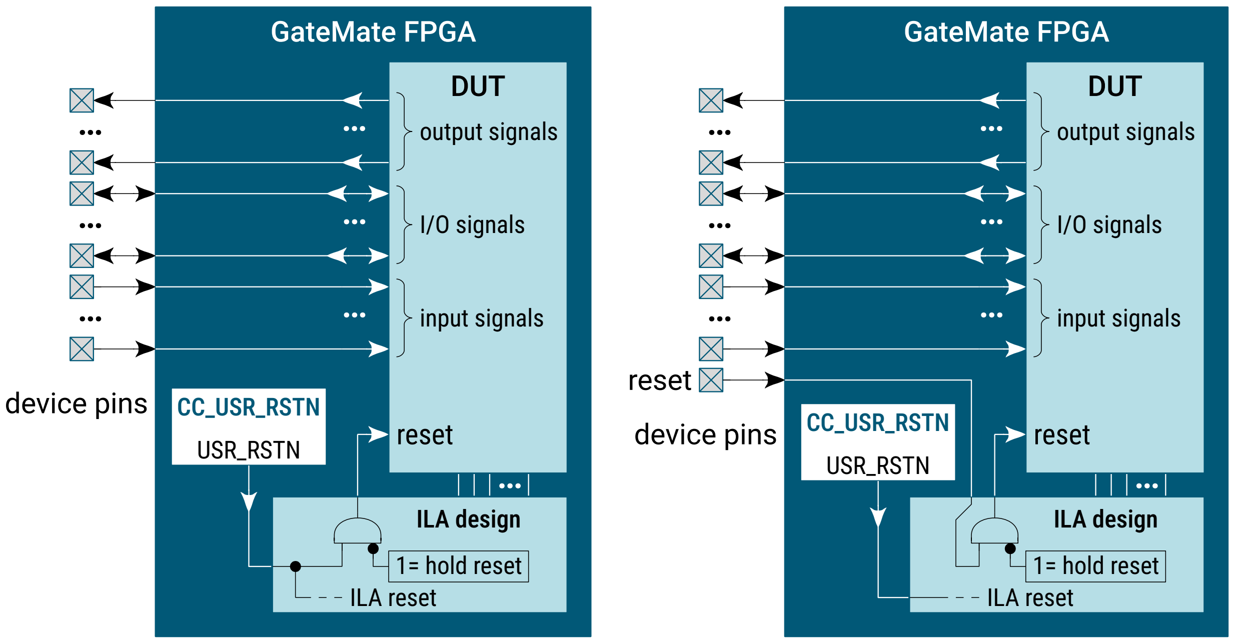

Figure 6: *DUT reset with reset control function (left: reset via primitive, right: reset via input pin)*

To resolve this problem, the user can enable the reset control function during configuration with the interactive shell. This allows the user to reset the DUT during runtime. ILA remains active during this process. The user can then terminate the DUT reset state in the menu. The block diagram of the reset control function is shown in Figure 6. The user can manually start the DUT operation via the menu of the interactive shell (start capture). This function allows the user to analyze the startup process of the DUT immediately after a reset.

Please note that ILA treats the reset signal as active LOW.

4 ILAcop.py Parameters and Options

The control program ILAcop.py is used to configure and execute the ILA design with a DUT. The script is stored in the app/ folder. Program options and parameters are shown with the following call:

```
$ > python3 ILAcop.py --help
```

```
usage: python3 ILAcop.py [Commands]

ILA version: 1.1.1

GateMate ILA control program.
With this script, you can configure and execute the ILA with a design under test (DUT
    ↪ ).

Commands:
  config    Configure the ILA.
              -vlog SOURCE     Paths to the Verilog source code files.
              -vhd SOURCE     Paths to the VHDL source code files.
              -t NAME              Top level entity of the design under test.
              -ccf SOURCE      Folder containing the .ccf file of the design under
                  ↪ test.
              -s SPEED             Configure ILA for best performance. Max Sample
                  ↪ Width = 40, the number of samples depends on the sample width.
              -f MHz                 Defines the external clock frequency in MHz (
                  ↪ default is 10.0 MHz).
              -sync LEVEL       Number of register levels via which the sampled
                  ↪ signals are synchronised (default: 2)
              -d DELAY            ILA PLL Phase shift of sampling frequency. 0=0°,
                  ↪ 1=90°, 2=180°, 3=270° (default: 2).
              -opt                      Optimizes the design by deleting all unused
                  ↪ signals before design evaluation.
          (optional) Subcommands config:
                -create_json: Creates a JSON file in which the logic analyzer can be
                    ↪ configured.
            NOTE: Without the subcommand the configurations are requested step by
                ↪ step via the terminal.

  reconfig  Configures the ILA based on a JSON file. With this option you have to
      ↪ specify a JSON file with -l [filename].json.

  start      Starts the communication to the ILA with the last uploaded config
               -s   The -s parameter prevents the FPGA from being reconfigured on
                  ↪ restart.

options:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
```

Common options (mode-independent):

`-h`, `--help`	Show this help message and exit.
`--version`	Show ILA version number and exit.
`--showdev`	Outputs all found FTDI devices including their USB ports and exit.
`--clean`	Delete all output files created by previous calls of the program in the directories `log/`, `net/`, `p_r_out/`, `app/vcd_files/`, `app/save_config/`, and `app/config_design/`.
`-wd WORK_DIR`	Folder from which Yosys will start the synthesis of the DUT. This option has only to be used if the default directory `../../bin/yosys/` does not fit.

Modes:

`config` Configures ILA via the interactive shell, creates the FPGA bitstream file (gateware), loads the FPGA with it and puts the DUT and ILA into operation. In addition, the interactive shell writes a JSON file with the settings. This can be used again later to restart the DUT and ILA without having to make the entries in the interactive shell again.

The JSON file can be edited to adjust settings manually and then read in later by ILA for setup (see mode `reconfig`).

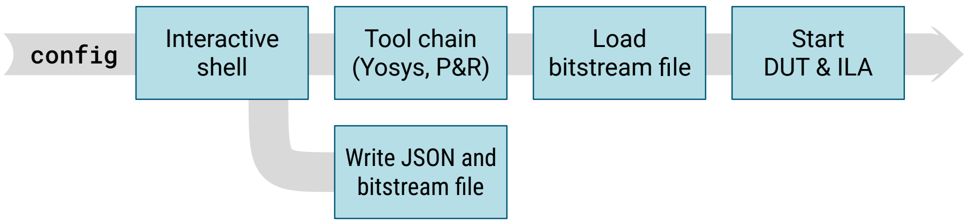

Figure 7: *ILA mode `config`*

`reconfig` Configures ILA based on a JSON file, whereby the user can change the recording and pre-trigger lengths. With this setting, a new FPGA bitstream file is generated and uploaded. The DUT and ILA are then put into operation.

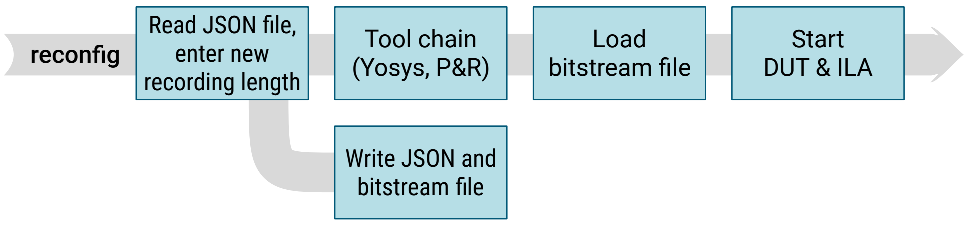
reconfig
Read JSON file, enter new recording length
Tool chain (Yosys, P&R)
Load bitstream file
Start DUT & ILA
Write JSON and bitstream file

Figure 8: *ILA mode `reconfig`*

`start` Configures ILA based on the last setup, optionally loads the FPGA bitstream file and puts the DUT and ILA into operation.

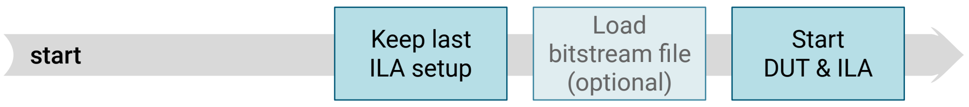

Figure 9: *ILA mode `start`*

Options for mode *config*:

`-vlog SOURCE`	Paths to the DUT `Verilog` source code files. Using the flag, provide one or more paths to directories containing the respective source code files. Multiple paths are separated by a space. The program will search for all `Verilog` files and build the design. Please note that subdirectories are not searched. This option can be combiend with `-vhd` option. If neither of the two options is specified, the programm exits with an error message.
`-vhd SOURCE`	Paths to the DUT VHDL source code files. Using the flag, provide one or more paths to directories containing the respective source code files. Multiple paths are separated by a space. The program will search for all VHDL files and build the design. Please note that subdirectories are not searched. This option can be combiend with `-vlog` option. If neither of the two options is specified, the programm exits with an error message.
`-t NAME`	Top level entity of the DUT. This option is required to build the design.
`-ccf SOURCE`	Path to the .ccf file of the DUT. This option is only required if the constraints file is not in the same folder as the given source code. All specified source code directories are searched.
`-f MHz`	Defines the external PLL reference frequency in MHz as a floating-point value (default is 10.0 MHz). This option is only relevant if the ILA design uses its own PLL for clock generation.
`-sync LEVEL`	Number of register levels via which the sampled DUT signals are synchronized. This option is used if the sampled signals are not synchronized to the ILA clock (default: 2).
`-d DELAY`	Set PLL phase shift of ILA sampling frequency: `0` = 0°, `1` = 90°, `2` = 180°(default) and `3` = 270°. Sampling always takes place on the rising edge. In most cases, the default setting is the best value. If there are timing problems, the sampling phase can be changed. If all sampled signals are generated on the same edge of a DUT clock and this is also the ILA clock frequency, then it usually makes sense to capture data in phase opposition, i.e. 180°. If the sampled signals are generated on both the falling and the rising clock edge, a phase setting of 90°or 270°can lead to better results. The optimum setting depends on the DUT and the sampled signals. This option is only relevant if the ILA design uses its own PLL for clock generation.
`-opt`	Optimizes the design by deleting all unused signals before design evaluation. This option should only be used if it does not eliminate any signals during synthesis that are required for troubleshooting with ILA. If this happens, the DUT will contain unused signals that have been selected for observation by ILA. In this case, the user should check the DUT implementation for possible design errors.

`-create_json:`	This option generates a JSON file without running the interactive shell completely. Furthermore, no bitstream file is generated. Nevertheless, ILAcop.py analyzes the DUT to write project-related content into the JSON file after the user has entered the ILA clock source and the reset function. The user must then subsequently fill the file with suitable settings. This JSON file can later be used with the `reconfig` mode to let Yosys generate the bitstream file and run the DUT and the ILA design.
`-s`	This option switches to fast ILA mode (expert setting). It is only used in special cases when ILA needs to capture particularly quickly. Only one block RAM is used, which greatly reduces the number of data values that can be recorded, but results in a compact ILA circuit and fast block RAM access.

Option for mode *reconfig*:

`-l FILE`	Without this parameter, the last written JSON file is used to set up ILAcop.py. A different JSON file can be selected with this option.

Option for mode *start*:

`-s`	The FPGA is normally reloaded in mode `start` with the existing bitstream file. This can be prevented with this option. Instead, the FPGA retains its last loaded bitstream file.

Examples of use:

```
$ > python3 ILAcop.py config -vlog ..\textbackslash example\_dut\
   ↪ textbackslash blink\textbackslash src\textbackslash ~-t blink
```

Calls the interactive shell for ILA setup specification, automatically runs the toolchain with the specified path for `Verilog` source files and top level module `blink`, creates and loads the gateware, finally starts the DUT and ILA design.

```
  $ > python3 ILAcop.py reconfig -l ila\_config\_blink\_2025\_03\_30\_13\
     ↪ _02\_00.json
```

Reads the specified JSON file, loads the gateware, finally starts the DUT and ILA design.

```
  $ > python3 ILAcop.py start
```

Loads the gateware, starts the DUT and ILA design while retaining the last ILA settings.

5 ILA Setup File `config.py`

5.1 Common Remarks

The setup file `config.py` is already introduced in Chapter 2.5. All permitted entries are presented in detail in this chapter.

General notes on the file syntax:

- The basic syntax is "`<parameter> = <value>`".
- Character strings should be enclosed in single quotation marks. This is necessary if they contain spaces or special characters.
- Integer values must not be enclosed in quotation marks.
- Parameters and values are case sensitive.
- For paths, slash must be used for Linux. For Windows, either slash or backslash can be used.
- Comment up to the end of the line is introduced with the hash character `#`.

5.2 Synthesis Tool and Calling Parameters

The executable synthesis file and its parameters can be set as follows:

```
YOSYS       = 'yosys'
YOSYS_FLAGS = '-nomx8'
```

If the operating system does not know the path to the executable, it must be entered in the configuration file, e.g.

```
YOSYS = '/home/dave/cc-toolchain-linux/bin/yosys/yosys'
```

The parameters must be set according to the requirements of the user. The default setting is `'-nomx8'` to exclude the use of `CC_MX8` multiplexer cells in the output netlist.

Please see the Yosys documentation for more information.

5.3 Place & Route Tool and Calling Parameters

The Cologne Chip Place & Route executable file can be downloaded from this link.

```
PR       = 'p_r'
PR_FLAGS = '-cCP +uCIO'
```

If the operating system does not know the path to the executable, it must be entered in the configuration file, e.g.

```
PR = '/home/dave/cc-toolchain-linux/bin/p_r/p_r'
```

The `'-cCF'` flag specifies that a .ccf file is to be used. This file is automatically appended by the program and must therefore not be specified in the setup file.

The `'+uCIO'` flag must not be removed. It switches the configuration GPIO bank of the FPGA so that these can be used as normal user IOs after configuration. This is necessary for the ILA to be able to communicate via the same connection as it was configured. The flag can only be omitted if other IOs are selected for the SPI communication of the ILA design.

Please see the GateMate™ FPGA Datasheet for more information concerning the Place & Route options

5.4 FPGA Bitstream Upload

The software for the configuration of the GateMate FPGA, as well as the respective programming mode, can be customised under the following parameters:

```
UPLOAD       = 'openFPGALoader'
UPLOAD_FLAGS = '-b gatemate_evb_jtag'
```

Other programs to load the bitstream file into the FPGA can of course also be specified. If the operating system does not know the path to the executable, it must be entered in the configuration file, e.g.

```
UPLOAD       = '/home/dave/cc-toolchain-linux/bin/openFPGALoader/
   ↪ openFPGALoader'
```

The example above uses the GateMate™ FPGA Evaluation Board. Please note, that the correct programming mode must be set on the board. The configuration file is defaulted to the GateMate™ FPGA Evaluation Board and configures it directly via the JTAG interface.

In the following example, again the GateMate™ FPGA Evaluation Board is used and the ILA design communicates with the ILAcop.py via SPI in passive mode:

```
UPLOAD_FLAGS = '-b gatemate_evb_spi'
```

Alternatively, other settings of `UPLOAD_FLAGS` can be selected, namely one of the following:

```
UPLOAD_FLAGS = '-b gatemate_pgm_spi'
```

```
UPLOAD_FLAGS = '-b gatemate_pgm_jtag'
```

Finally, the Olimex evaluation board [4] can be used as follows:

```
UPLOAD_FLAGS = '-b olimex_gatemateevb'
```

Additional options are available, as shown in the following example:

```
UPLOAD_FLAGS = '-b gatemate_evb_spi -f --verify'
```

`-f` does not load the bitfile into the FPGA, but writes it into the on-board flash memory.

If this function is used, `--verify` can also be entered to verify that the bitfile has correctly been stored in the flash memory.

Please see the openFPGALoader documentation for more information about the upload options.

5.5 Waveform Viewer

Any waveform viewer can be used with ILA, that can read a .vcd file, e.g.

```
REPRESENTATION_SOFTWARE = ['gtkwave']
REPRESENTATION_FLAGS    = ['--save', 'save.gtkw']
```

All arguments for `GTKWave` must be stored in an array. The `'--save'` flag specifies a file in which the view configurations made in `GTKWave` will be stored.

If the operating system does not know the path to the executable, it must be entered in the configuration file, e.g.

```
REPRESENTATION_SOFTWARE = '/home/dave/gtkwave/bin/gtkwave'
```

Please note that the software places the .vcd file name between `REPRESENTATION_SOFTWARE` and `REPRESENTATION_FLAGS` when it is called.

5.6 ILA Connection to External Hardware

In addition to the `UPLOAD_FLAGS` parameter already described in Section 5.4, ILA requires further information on how the connection to the FPGA is established.

```
UPLOAD_FLAGS = '-c gatemate_pgm'
CON_DEVICE = 'pgm'
```

Different values can be specified for `CON_DEVICE` depending on the hardware used:

'pgm': GateMate™ FPGA Programmer is used in JTAG or SPI mode (all versions, regardless of whether it has activatable level shifters).

[4] Open Source SDR Lab Kintex-7 325t FPGA PCIE Development Board

'evb': GateMate^{TM} FPGA Evaluation Board is used in JTAG or SPI mode (all versions, regardless of whether it has activatable level shifters).

'oli': Olimex evaluation board is used (all versions, regardless of whether it has activatable level shifters).

'cust': Freely configurable mode with activatable level shifters. With this option, the GPIOs must be set appropriately with the parameters `cust_*` before the FTDI chip is addressed (see below).

'free': Freely customizable mode without activatable level shifters.

If the specific user hardware uses level shifters between the FTDI chip and the GateMate FPGA, the following parameters must also be adjusted:

```
CON_DEVICE = 'cust'
cust_gpio_direction_pins = 0x17F0
cust_gpio_direction = 0x1710
cust_gpio_write = 0x0210
```

If the GateMate^{TM} FPGA Programmer is used, the FTDI chip must be specified as follows:

```
CON_DEVICE = 'pgm'
CON_LINK = 'ftdi://ftdi:232h/1'
```

For the GateMate^{TM} FPGA Evaluation Board, a different setting is necessary:

```
CON_DEVICE = 'evb'
CON_LINK = 'ftdi://ftdi:2232h/1'
```

The Olimex evaluation board does not need this parameter.

Any other user hardware with activatable level shifters must be set up according to the used FTDI interface chip.

5.7 SPI Frequency

ILAcop.py communicates with the FPGA via an SPI interface. The SPI clock frequency is preset to 20 MHz. To use a different clock frequency, for example 10 MHz, enter the following line in `config.py`:

```
freq_max = 10000000
```

5.8 Block RAM

The FPGA CCGM1A1 has a total of 32 block RAMss. The ILA design can use all free block RAMs for data capturing that are not occupied by the DUT.

For stability reasons, the actual number of block RAMs should not be made available for the DUT and the ILA design. The default setting is 30 block RAMs:

```
available_BRAM = 30 # 40k RAM blocks
```

This value should not be increased.

However, a smaller value may be set to reduce the length of the data recording if this is desired.

Based on the given number of signals to be analyzed and the available number of block RAMs, the program automatically calculates how FIFOs can be cascaded in parallel and in series to achieve different numbers of data samples to be stored. The user can then choose between various numbers of signals to analyze (see Section 8.3 from page 37). The program displays the duration time in microseconds, the total data sample count, the constructed FIFO cascade, as well as the width and depth of the individual FIFOs used (see Section 3.4).

6 Hardware Setup

The GateMate^{TM} FPGA Evaluation Board or customer hardware can be used for ILA. In either case, the computer must be connected to the FPGA hardware via the USB interface, as shown in Figure 10. The connection to the FPGA is then made either via the GateMate's SPI or JTAG interface.

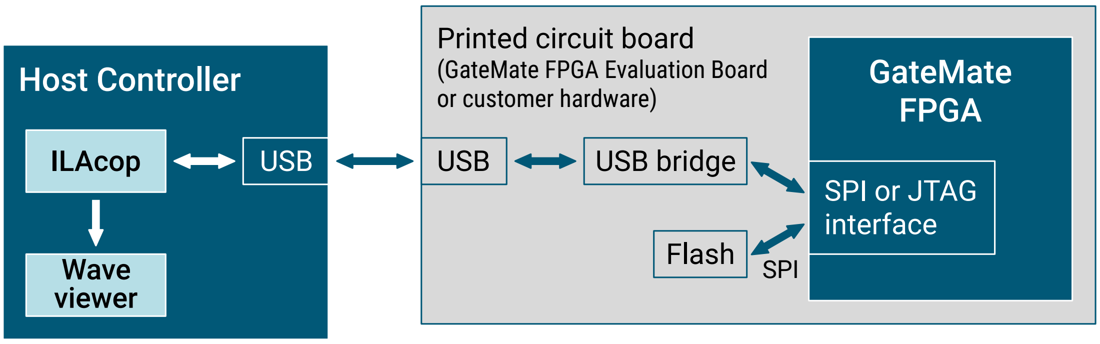

Figure 10: *ILA hardware setup*

The FPGA datastream can be loaded from the flash memory or it can be transferred directly from the computer. To do this, the correct configuration mode must be set for the device, as documented in the GateMate^{TM} FPGA CCGM1A1 Datasheet or the GateMate^{TM} FPGA Evaluation Board Datasheet.

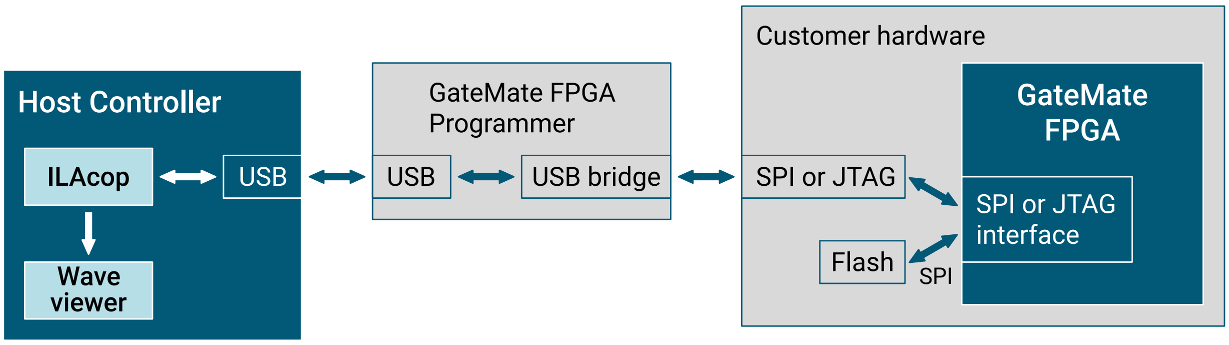

Figure 11: *ILA hardware setup using the GateMate^{TM} FPGA Programmer*

If a customer printed circuit board is used, the USB bridge does not have to be part of this PCB. Instead, the GateMate^{TM} FPGA Programmer Board Datasheet can be used and connected to a JTAG or SPI interface of the customer PCB as shown in Figure 11.

In any case, make sure that the program has access rights to the USB port to which the FPGA is connected. This is done with a call

```
python3 .\ILAcop.py --showdev
```

which shows all available FTDI interfaces, e.g.

```
Available interfaces:
ftdi://ftdi:2232:E1-31B0220/1   (GateMate FPGA Evalboard 3.2)
ftdi://ftdi:2232:E1-31B0220/2   (GateMate FPGA Evalboard 3.2)
```

In this example, the dual port USB bridge FT2232 of the GateMate^{TM} FPGA Evaluation Board is displayed.

7 ILA SPI Interface

When ILAcop.py loads the bitstream file into the FPGA via the SPI interface, the SPI pins of the configuration controller are involved. Later, when the DUT is in operation and ILAcop.py controls the data capturing, this is done via the SPI interface, too. But there are a few differences to note, as shown in Figure 12:

DUT setup: The bitstream can be loaded via the JTAG or SPI interface of the GateMate FPGA. In both cases the configuration controller is involved and therefore the pins of the interface are fixed. In the case of the SPI interface, the FPGA operates in SPI active mode and ILAcop.py in SPI passive mode.

DUT operation: ILAcop.py always controls data capturing via an SPI interface. Typically, and this is the default setting, the same FPGA pins are used as for the bitstream upload. It is also possible to connect the SPI interface of ILAcop.py to other GPIO pins of the FPGA. This is usually only useful if (a) the bitstream was loaded via JTAG and (b) the SPI interface of the FPGA configuration controller has an atypical use in the operation of the DUT. However, ILAcop.py is now in SPI active mode and the FPGA in SPI passive mode.

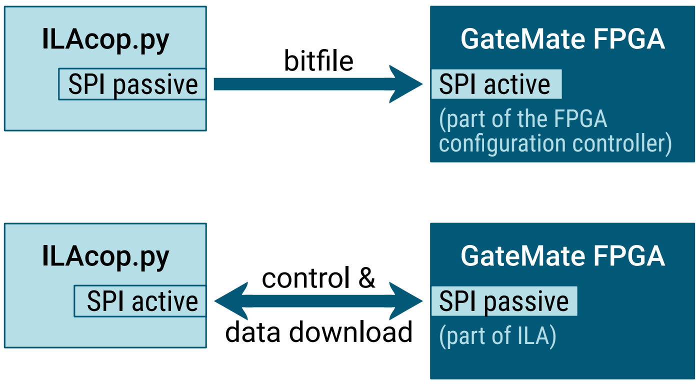

Figure 12: *SPI interface of ILAcop.py*

The pins used by the SPI-passive controller of the gateware to communicate with the ILAcop.py can be adjusted in the file `src/ILA_top.ccf`. They are located by default at the following pins:

```
Pin_in   "i_sclk_ILA"   Loc = "IO_WA_A5" | SCHMITT_TRIGGER = true;
Pin_in   "i_mosi_ILA"   Loc = "IO_WA_A4";
Pin_out  "o_miso_ILA"   Loc = "IO_WA_B3";
Pin_in   "i_ss_ILA"     Loc = "IO_WA_B4";
```

8 Use of ILA

8.1 Workflow

The ILA workflow is shown in Figure 13. In addition to the ILA design, the user design to be examined, called *device under test* (DUT), is of course also required. The procedure roughly consists of the following steps which are all grouped together in ILAcop.py:

1. Execution of Yosys to analyze the DUT signals.

   A list of all DUT signals is generated.

2. Configuration of ILA in an interactive Shell.

   Every time ILA was configured with the interactive shell, the configuration is stored in a JSON file. The file can be edited and ILA can be executed again at a later time with the configurations defined in the file. For more informations see Section 8.4.

3. Another call of Yosys to read both the user design and the ILA design for synthesis.

   After the user has configured ILA, the entire design consisting of the user design and the ILA design is synthesized.

4. Execution of Place & Route to generate the FPGA gateware (bitstream file).

   The output file will be stored in `pr_out/`.

5. Configuration of the GateMate FPGA.

   Then the toolchain will be executed to bring the DUT together with the ILA design on the FPGA. If this has been successfully completed, the communication to the ILA design on the FPGA is automatically started.

6. Operation of the FPGA circuitry and analysis of the signals according to the previous ILA configuration.

   Once the DUT has been put into operation together with the ILA design, trigger monitoring, data capturing and downloading run automatically. The user can change the trigger condition during operation if desired.

7. Examination of the exported wave form.

   Once data recording is complete, the stored data is automatically downloaded to the computer into the directory `app/vcd_files/` and displayed in the wave viewer.

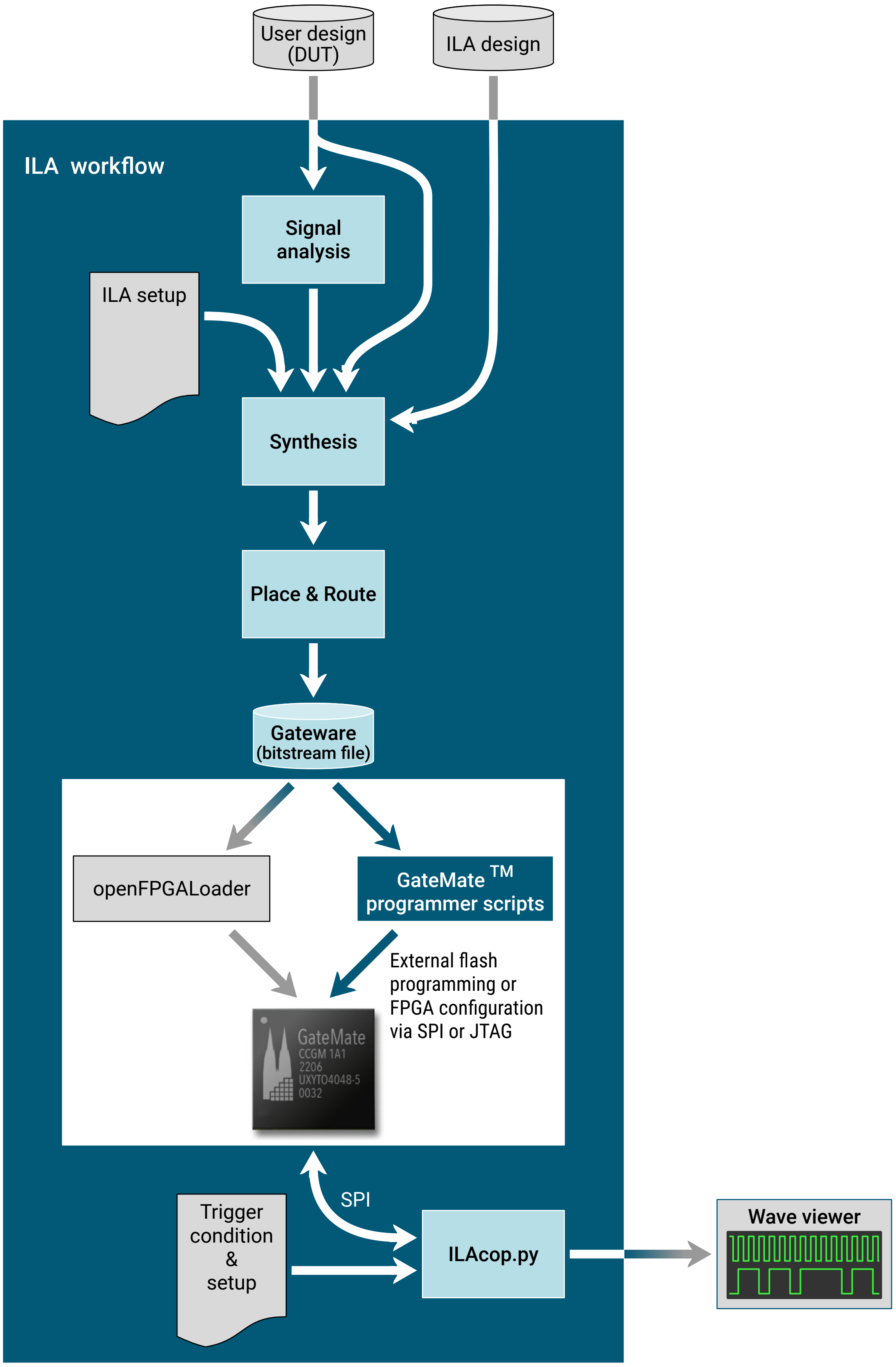
User design (DUT)
ILA design
ILA workflow
Signal analysis
ILA setup
Synthesis
Place & Route
Gateware (bitstream file)
openFPGALoader
GateMate TM programmer scripts
External flash programming or FPGA configuration via SPI or JTAG
GateMate CCGM 1A1 2206 UXYT04048-5 0032
SPI
Trigger condition & setup
ILAcop.py
Wave viewer

Figure 13: *Overview of the ILA workflow*

8.2 The ILAcop.py Menu

After the gateware has been loaded into the FPGA, ILAcop.py displays the menu and waits for the user's selection. The following functions can be executed:

```
0 —— exit
1 —— change Trigger
2 —— start capture
3 —— reset ILA (resets the config of the ILA)
4 —— reset DUT (hold the DUT in reset until the capture starts)

Enter your choice: 1
```

`0 – exit`
The interactive shell ILAcop.py is terminated. The DUT continues to run. The interactive shell can be restarted with

```
ILAcop start -s
```

without interrupting the running DUT process.

`1 – change trigger`
The ILA design and DUT are stopped. The interactive shell then asks the user again for the trigger condition. Either a single trigger event or a trigger sequence can be specified. After this, a new bitstream file is generated and loaded into the FPGA. ILAcop.py then presents the menu again and the user can continue with the analysis of the DUT, which is usually done with option 2.

`2 – start capture`
The ILA design waits for the trigger event and then starts data capturing. While ILA is waiting for the trigger event, the user can cancel the process at any time by pressing the ENTER key. After the end of data capturing, the data is automatically downloaded to the computer and displayed in the specified wave viewer. ILAcop.py then presents the menu again so that the user can continue with the DUT analysis.

`3 – reset ILA (resets the configuration of the ILA)`
The trigger event is reset. The ILA design then waits again for the trigger event to restart data capturing.

`4 – reset DUT (hold the DUT in reset until the capture starts)`
This function can only be used if the reset control function is activated (see Section 3.6 on page 18). If option 4 is selected, the DUT reset is activated and remains active until the user either enters option 4 again to end the reset or option 2 to start the wait for the trigger event and then perform the data capturing.

8.3 Configuration Example with the Interactive Shell

This section provides step-by-step instructions on how to configure ILA with the interactive shell using the Game of Life (GoL) example design `ws2812_gol.vhd` [5].

To visualize the status of the GoL matrix, an 8×8 `WS2812` LED element is connected to the GateMate™ FPGA Evaluation Board. This receives the three bytes of the RGB value for each of the 64 LEDs in turn via a serial interface. For this purpose, the states of the matrix are temporarily stored in the internal RAM. Another process reads the data, serializes it and then outputs it in a `WS2812`-compliant manner.

It is not necessary to actually have the `WS2812` LED matrix connected to the evaluation board. The GoL circuitry example and the demonstration of the ILA will of course also work without this output unit.

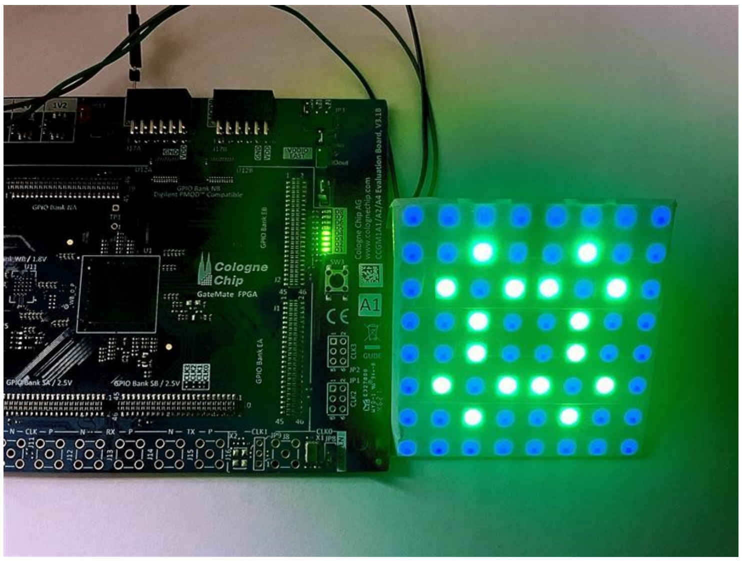

Figure 14: *GateMate™ FPGA Evaluation Board with WS2812 LED matrix*

Open a console, change to the directory `app/` of the ILA installation and call ILAcop.py [6]:

```
gatemate_ila\app> python3 ILAcop.py config -vhd ..\example_DUT\ws2812_gol
   ↪ \src\ -t ws2812_gol
```

[5] `ws2812_gol` implements the Conway's Game of Life (GoL) simulation on an 8×8 matrix. Each cell of the matrix has either the state 'dead' or 'alive' and it has eight neighbors. Cells at the edges have virtual neighbors outside the matrix that are basically dead. The initial state of the matrix is generated at random.

- A dead cell becomes alive when it has exactly three living neighbors.
- A living cell becomes dead if it has more than three or less than two living neighbors.

With the `gol_next_gen` signal, all cells calculate their next state simultaneously, depending on the neighboring states that are present at the time of the clock edge. The next state is then output synchronously within one clock cycle.

[6] ILAcop.py is a console-only application.

The path to the top level entity `ws2812_gol` of the GoL design may need to be adjusted.

After calling ILAcop.py, the program first outputs the names of the .ccf file and the Very High Speed Integrated Circuit Hardware Description Language (VHDL) source files of the DUT. Then Yosys is called to prepare the design for ILA, which extract all DUT signals of the modules, among other things, afterwards.

Since ILAcop.py uses the block RAMs not occupied from the DUT for data sample storage, the block RAMs used by the DUT are also analyzed and output.

```
################# ccf File ##################
#                                           #
# ws2812_gol.ccf                            #
#                                           #
#############################################

################ vhdl Files #################
#                                           #
# aserial.vhd                               #
# edge_detection.vhd                        #
# gol_8x8_control.vhd                       #
# gol_cell.vhd                              #
# init_package.vhd                          #
# ram.vhd                                   #
# ram_to_bit.vhd                            #
# receive_command.vhd                       #
# spi_slave.vhd                             #
# ws2812_gol.vhd                            #
#                                           #
#############################################

Examine DUT ...

############# Block RAM in use ##############
#                                           #
# CC_BRAM_20K in use: 1                     #
# CC_BRAM_40K in use: 0                     #
#                                           #
#############################################

!!!!!!!!!!!!!!!!!!!!!!!!!!!! NOTE !!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                            !
! Now you will be guided through the configuration of the ILA. !
! Entering 'e' exits the process and generates a configurable !
! JSON file for the given DUT.                               !
! Enter 'p' for 'previous' to backtrack a step.              !
!                                                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

ILA only offers clock sources (see Section 3.3) that are also available in the DUT, supplemented by the option for the user to select a separate PLLfor ILA and set it to the desired frequency.

In this example, the DUT does not use a PLL, which is why only 2 options are offered below:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! NOTE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                         !
! In the following, a clock source for the ILA should be selected.       !
! Usually, the same clk signal that clocks the tested signals suffices.  !
!                                                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Here are the possible ways to provide a clock to the ILA:

 1 = Use an external clk input signal.
 2 = Use an additional PLL with a freely selectable frequency (additional net of the
     ↪ global Mesh are required).

please choose between 1 and 2: 1

########### found DUT clk source ############
#                                          #
# Input serves as ILA clk source: "clk"    #
#                                          #
############################################

Do you want to change the clk source? (y:yes/N:no): N
```

The clock source is accepted and now reset settings have to be specified. ILAcop.py determines the reset signal, outputs it and asks the user whether the selection should be accepted or another reset signal should be selected:

```
!!!!!!!!!!!!!!!!!!!! User controllable reset !!!!!!!!!!!!!!!!!!!
!                                                               !
! The ILA can hold the DUT in reset until capture starts.       !
! This makes it possible to capture the start process of the DUT !
! Attention, the ila treats the signal as active LOW            !
!                                                               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

The following options are available:

 1 = Use an external reset input signal. Potential input found: reset
 2 = Deactivate this function.

please choose between 1 and 2: 1

Would you like to choose a different user controllable input than 'reset'? (y:yes/N:
    ↪ no): N
```

ILAcop.py now outputs all DUT signals found. The signals are listed in a table, including their bit width and the module name in which they were found. The table also shows whether a signal has already been selected for analysis. It is possible to filter the signals by module, e.g. to display only the signals of a specific module. This is particularly useful for large designs.

In addition, the maximum possible number of bits to be analyzed and the number of bits currently selected are displayed. If this is exceeded, a corresponding warning appears.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! NOTE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                              !
! You will be prompted to select signals for analysis from those found in your !
! design under test.                                                           !
!                                                                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

---------------------------------- ws2812_gol ----------------------------------
+------+-----------------------+---------+----------+--------------------------+
|    # | name                  | range   | selected | hierarchy                |
+------+-----------------------+---------+----------+--------------------------+
|    1 | byte_receive          | [7:0]   |    []    |                          |
|    2 | byte_send             | [7:0]   |    []    |                          |
|    3 | clk                   |   1     |    []    |                          |
...
|   38 | gol_init              |   1     |    []    | \golx64.                 |
|   39 | gol_next_gen          |   1     |    []    | \golx64.                 |
|   40 | init_pattern          | [63:0]  |    []    | \golx64.                 |
|   41 | life_out              | [99:0]  |    []    | \golx64.                 |
...
|   46 | rgb_color             | [1:0]   |    []    | \golx64.                 |
|   47 | shift_life_row        | [7:0]   |    []    | \golx64.                 |
...
|   59 | ws2812_ram_addr_wr    | [7:0]   |    []    | \golx64.                 |
...
|   62 | ws2812_rgb_byte_reg   | [7:0]   |    []    | \golx64.                 |
...
| 1166 | shift_rgb_byte        | [7:0]   |    []    | \ram_to_bit.             |
...
| 1171 | ws2812_ram_addr_rd    | [7:0]   |    []    | \ram_to_bit.             |
...
| 1175 | data_out              |   1     |    []    | \ram_to_bit.aserial.     |
...
| 1189 | start_ram_to_bit      |   1     |    []    |                          |
...
| 1210 | start_1               |   1     |    []    | \write_bram_rec_cmd.     |
| 1211 | start_2               |   1     |    []    | \write_bram_rec_cmd.     |
+------+-----------------------+---------+----------+--------------------------+

## Number of selected bits to be analysed ###
#                                          #
# 0 (max. 1180)                            #
#                                          #
############################################

Select signals to be analyzed (0 = finish, f = filter): f
```

(For a better overview, the list is abbreviated here.)

The user can now either enter the number of a signal to include it in the signal vector, or he can shorten the signal list with a module filter or he can end the signal selection procedure with the input ‘`0`’.

When selecting a signal bus, the user can select individual bits, a range, or even several individual bits and ranges for the signal.

If a signal that has already been choosen is selected again, the selection is deselected (toggle function).

After each selected signal, the list of signals is displayed again. The previously selected signals are entered in the ‘selected’ column, where `['A']` means that the entire signal bus has been selected (‘all’).

The user is then prompted to enter another signal, to change the module filter or to end the signal selection procedure.

If the module filter is used, only the signals of the selected module are displayed. The selection of a module looks as follows:

```
————————————————————— DUT moduls —————————————————————
+----+------------------------------------------------+
|  # | moduls                                         |
+----+------------------------------------------------+
|  0 | ws2812_gol.                                    |
|  1 | ws2812_gol.\dualportram.                       |
|  2 | ws2812_gol.\golx64.                            |
|  3 | ws2812_gol.\golx64.gol_row:1.gol_column:1.gol. |
|  4 | ws2812_gol.\golx64.gol_row:1.gol_column:2.gol. |
...
| 65 | ws2812_gol.\golx64.gol_row:8.gol_column:7.gol. |
| 66 | ws2812_gol.\golx64.gol_row:8.gol_column:8.gol. |
| 67 | ws2812_gol.\ram_to_bit.                        |
| 68 | ws2812_gol.\ram_to_bit.addrcnt.                |
| 69 | ws2812_gol.\ram_to_bit.aserial.                |
| 70 | ws2812_gol.\ram_to_bit.shift_cnt.              |
| 71 | ws2812_gol.\spi_edge_detect.                   |
| 72 | ws2812_gol.\spi_slave.                         |
| 73 | ws2812_gol.\write_bram_rec_cmd.                |
+----+------------------------------------------------+

Select a module from which you would like to analyze signals: 2
```

```
──────────────── \golx64. signals ────────────────
+────+───────────────────────+─────────+──────────+
|  # | name                  |  range  | selected |
+────+───────────────────────+─────────+──────────+
|  1 | break_counter         |  [23:0] |    []    |
|  2 | clk                   |    1    |    []    |
|  3 | counter_index         |  [3:0]  |    []    |
|  4 | din_spi               |  [7:0]  |    []    |
|  5 | gol_init              |    1    |    []    |
|  6 | gol_next_gen          |    1    |    []    |
|  7 | init_pattern          |  [63:0] |    []    |
|  8 | life_out              |  [99:0] |    []    |
|  9 | life_shift_cnt        |  [2:0]  |    []    |
| 10 | nachbarn              |  [35:0] |    []    |
| 11 | neighbours_in         | [511:0] |    []    |
| 12 | reset                 |    1    |    []    |
| 13 | rgb_color             |  [1:0]  |    []    |
| 14 | shift_life_row        |  [7:0]  |    []    |
| 15 | start_sm_new          |    1    |    []    |
| 16 | state_color_to_ram    |  [2:0]  |    []    |
| 17 | tl_gol_state          |  [3:0]  |    []    |
| 18 | waddr_spi             |  [2:0]  |    []    |
| 19 | write_en              |    1    |    []    |
| 20 | write_en_ram          |    1    |    []    |
| 21 | write_en_s            |    1    |    []    |
| 22 | write_ws2812_out      |    1    |    []    |
| 23 | write_ws2812_s        |    1    |    []    |
| 24 | writeram              |    1    |    []    |
| 25 | ws2812_busy           |    1    |    []    |
| 26 | ws2812_ram_addr_wr    |  [7:0]  |    []    |
| 27 | ws2812_ram_addr_wr_s  |  [7:0]  |    []    |
| 28 | ws2812_rgb_byte       |  [7:0]  |    []    |
| 29 | ws2812_rgb_byte_reg   |  [7:0]  |    []    |
+────+───────────────────────+─────────+──────────+

## Number of selected bits to be analysed ###
#                                           #
# 0 (max. 1180)                             #
#                                           #
#############################################

Select signals to be analyzed (0 = finish, f = no filter, c = change filter): 5
```

The signals of the selected module can now be entered in the list for analysis:

```
───────────────── \golx64. signals ─────────────────
+────+────────────────────────+──────────+───────────+
|  # | name                   |  range   | selected  |
+────+────────────────────────+──────────+───────────+
|  1 | break_counter          |  [23:0]  |    []     |
|  2 | clk                    |    1     |    []     |
|  3 | counter_index          |  [3:0]   |    []     |
|  4 | din_spi                |  [7:0]   |    []     |
|  5 | gol_init               |    1     |  ['A']    |
|  6 | gol_next_gen           |    1     |    []     |
|  7 | init_pattern           |  [63:0]  |    []     |
|  8 | life_out               |  [99:0]  |    []     |
|  9 | life_shift_cnt         |  [2:0]   |    []     |
| 10 | nachbarn               |  [35:0]  |    []     |
| 11 | neighbours_in          | [511:0]  |    []     |
| 12 | reset                  |    1     |    []     |
| 13 | rgb_color              |  [1:0]   |    []     |
| 14 | shift_life_row         |  [7:0]   |    []     |
| 15 | start_sm_new           |    1     |    []     |
| 16 | state_color_to_ram     |  [2:0]   |    []     |
| 17 | tl_gol_state           |  [3:0]   |    []     |
| 18 | waddr_spi              |  [2:0]   |    []     |
| 19 | write_en               |    1     |    []     |
| 20 | write_en_ram           |    1     |    []     |
| 21 | write_en_s             |    1     |    []     |
| 22 | write_ws2812_out       |    1     |    []     |
| 23 | write_ws2812_s         |    1     |    []     |
| 24 | writeram               |    1     |    []     |
| 25 | ws2812_busy            |    1     |    []     |
| 26 | ws2812_ram_addr_wr     |  [7:0]   |    []     |
| 27 | ws2812_ram_addr_wr_s   |  [7:0]   |    []     |
| 28 | ws2812_rgb_byte        |  [7:0]   |    []     |
| 29 | ws2812_rgb_byte_reg    |  [7:0]   |    []     |
+────+────────────────────────+──────────+───────────+

## Number of selected bits to be analysed ###
#                                          #
# 1 (max. 1180)                            #
#                                          #
############################################

 Select signals to be analyzed (0 = finish, f = no filter, c = change filter): 6
```

This step is repeated until all the desired signals have been selected.

```
─────────────── \golx64. signals ───────────────
+────+──────────────────────+─────────+──────────+
|  # | name                 |  range  | selected |
+────+──────────────────────+─────────+──────────+
|  1 | break_counter        |  [23:0] |    []    |
|  2 | clk                  |    1    |    []    |
|  3 | counter_index        |  [3:0]  |    []    |
|  4 | din_spi              |  [7:0]  |    []    |
|  5 | gol_init             |    1    |  ['A']   |
|  6 | gol_next_gen         |    1    |  ['A']   |
|  7 | init_pattern         |  [63:0] |    []    |
|  8 | life_out             |  [99:0] |    []    |
|  9 | life_shift_cnt       |  [2:0]  |    []    |
| 10 | nachbarn             |  [35:0] |    []    |
| 11 | neighbours_in        | [511:0] |    []    |
| 12 | reset                |    1    |    []    |
| 13 | rgb_color            |  [1:0]  |    []    |
| 14 | shift_life_row       |  [7:0]  |    []    |
| 15 | start_sm_new         |    1    |    []    |
| 16 | state_color_to_ram   |  [2:0]  |    []    |
| 17 | tl_gol_state         |  [3:0]  |    []    |
| 18 | waddr_spi            |  [2:0]  |    []    |
| 19 | write_en             |    1    |    []    |
| 20 | write_en_ram         |    1    |    []    |
| 21 | write_en_s           |    1    |    []    |
| 22 | write_ws2812_out     |    1    |    []    |
| 23 | write_ws2812_s       |    1    |    []    |
| 24 | writeram             |    1    |    []    |
| 25 | ws2812_busy          |    1    |    []    |
| 26 | ws2812_ram_addr_wr   |  [7:0]  |    []    |
| 27 | ws2812_ram_addr_wr_s |  [7:0]  |    []    |
| 28 | ws2812_rgb_byte      |  [7:0]  |    []    |
| 29 | ws2812_rgb_byte_reg  |  [7:0]  |    []    |
+────+──────────────────────+─────────+──────────+

## Number of selected bits to be analysed ###
#                                           #
# 2 (max. 1180)                             #
#                                           #
#############################################

Select signals to be analyzed (0 = finish, f = no filter, c = change filter): 8
```

The selection of a signal vector also requires the specification of which bits of the vector are to be analyzed.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! NOTE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                                  !
! Define a range for the vector to be analyzed.                                    !
!  you can do this in the following ways:                                          !
!   1) Press enter to analyze the entire vector                                    !
!   2) Define an area of the vector. (The area should be within the vector area):  !
!        e.g.: '[1:0]'                                                             !
!   3) Individual signals:                                                         !
!        e.g.: '1'                                                                 !
!   4) Any combination of areas and individual signals                             !
!        e.g.: '9, [7:5], 3, [1:0]'                                                !
! define Signals in descending order!                                              !
!                                                                                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

wire [99:0] life_out: 88:81, 78:71, 68:61, 58:51, 48:41, 38:31, 28:21, 18:11
```

```
-------------------------- \golx64. signals ---------------------------
+----+------------------------+--------+----------------------+
|  # | name                   | range  |       selected       |
+----+------------------------+--------+----------------------+
|  1 | break_counter          | [23:0] |          []          |
|  2 | clk                    |   1    |          []          |
|  3 | counter_index          | [3:0]  |          []          |
|  4 | din_spi                | [7:0]  |          []          |
|  5 | gol_init               |   1    |        ['A']         |
|  6 | gol_next_gen           |   1    |        ['A']         |
|  7 | init_pattern           | [63:0] |          []          |
|  8 | life_out               | [99:0] | ['88:81', '78:71',   |
|    |                        |        |  '68:61', '58:51',   |
|    |                        |        |  '48:41', '38:31',   |
|    |                        |        |  '28:21', '18:11']   |
|  9 | life_shift_cnt         | [2:0]  |          []          |
| 10 | nachbarn               | [35:0] |          []          |
| 11 | neighbours_in          | [511:0]|          []          |
| 12 | reset                  |   1    |          []          |
| 13 | rgb_color              | [1:0]  |          []          |
| 14 | shift_life_row         | [7:0]  |          []          |
| 15 | start_sm_new           |   1    |          []          |
| 16 | state_color_to_ram     | [2:0]  |          []          |
...
| 26 | ws2812_ram_addr_wr     | [7:0]  |          []          |
| 27 | ws2812_ram_addr_wr_s   | [7:0]  |          []          |
| 28 | ws2812_rgb_byte        | [7:0]  |          []          |
| 29 | ws2812_rgb_byte_reg    | [7:0]  |          []          |
+----+------------------------+--------+----------------------+

## Number of selected bits to be analysed ###
#                                          #
# 66 (max. 1180)                           #
#                                          #
############################################

Select signals to be analyzed (0 = finish, f = no filter, c = change filter): 13
```

(For a better overview, the list is abbreviated here.)

The user can now gradually select all the signals to be analyzed. This is not described in detail here.

Hint: For a large design with many signals, it may be easier to create a JSON file to configure ILA. In this way, any editor can be used to search for the signal to be analyzed. See Section 8.4 for more information on configuration via a JSON file.

Once all signals have been selected, the configuration process can be continued by entering `0`. In this example, the list of signals looks like this:

```
-------------------------------------------- ws2812_gol ---------------------------------------------
+------+--------------------------+---------+------------------------+-----------------------------+
|    # | name                     | range   |        selected        | hierarchy                   |
+------+--------------------------+---------+------------------------+-----------------------------+
|    1 | byte_receive             | [7:0]   |           []           |                             |
|    2 | byte_send                | [7:0]   |           []           |                             |
|    3 | clk                      |   1     |         ['A']          |                             |
...
|   38 | gol_init                 |   1     |         ['A']          | \golx64.                    |
|   39 | gol_next_gen             |   1     |         ['A']          | \golx64.                    |
|   40 | init_pattern             | [63:0]  |           []           | \golx64.                    |
|   41 | life_out                 | [99:0]  | ['88:81', '78:71',     | \golx64.                    |
|      |                          |         |  '68:61', '58:51',     |                             |
|      |                          |         |  '48:41', '38:31',     |                             |
|      |                          |         |  '28:21', '18:11']     |                             |
|   42 | life_shift_cnt           | [2:0]   |           []           | \golx64.                    |
...
|   46 | rgb_color                | [1:0]   |         ['A']          | \golx64.                    |
|   47 | shift_life_row           | [7:0]   |         ['A']          | \golx64.                    |
...
|   59 | ws2812_ram_addr_wr       | [7:0]   |         ['A']          | \golx64.                    |
...
|   62 | ws2812_rgb_byte_reg      | [7:0]   |         ['A']          | \golx64.                    |
...
| 1166 | shift_rgb_byte           | [7:0]   |         ['A']          | \ram_to_bit.                |
...
| 1171 | ws2812_ram_addr_rd       | [7:0]   |         ['A']          | \ram_to_bit.                |
...
| 1175 | data_out                 |   1     |         ['A']          | \ram_to_bit.aserial.        |
| 1176 | reset                    |   1     |           []           | \ram_to_bit.aserial.        |
| 1177 | run                      |   1     |         ['A']          | \ram_to_bit.aserial.        |
...
| 1210 | start_1                  |   1     |           []           | \write_bram_rec_cmd.        |
| 1211 | start_2                  |   1     |           []           | \write_bram_rec_cmd.        |
+------+--------------------------+---------+------------------------+-----------------------------+

## Number of selected bits to be analysed ###
#                                           #
# 110 (max. 1180)                           #
#                                           #
#############################################

Which signals should be analyzed (0 = finish)? 0
```

(For a better overview, the list is abbreviated here.)

The user is then prompted to enter the capturing duration.

Taking into account the `available_BRAM` setting (see Section 5.8), all block RAMs that are not occupied by the DUT are made available to the ILA design.

Only values that are possible due to the available memory are displayed here:

```
!!!!!!!!!!!!!!!!!! Note !!!!!!!!!!!!!!!!!!!
!                                         !
! The capture duration must be defined.   !
! The maximum duration depends on:        !
!  - available ram                        !
!  - width of the sample                  !
!  - sampling frequency                   !
!                                         !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

------Please choose one of the following durations: ------
+---+---------+---------------+--------------+-----------+
| # | smp_cnt | duration [us] | FIFO Cascade |      FIFO |
+---+---------+---------------+--------------+-----------+
| 1 |    4096 |         40.96 |        1 x 1 | 10 x 4096 |
| 2 |    8192 |         81.92 |        2 x 1 |  5 x 8192 |
| 3 |   12288 |        122.88 |        1 x 3 | 10 x 4096 |
| 4 |   16384 |        163.84 |        2 x 2 |  5 x 8192 |
| 5 |   20480 |         204.8 |        1 x 5 | 10 x 4096 |
| 6 |   24576 |        245.76 |        2 x 3 |  5 x 8192 |
| 7 |   32768 |        327.68 |        2 x 4 |  5 x 8192 |
| 8 |   40960 |         409.6 |        2 x 5 |  5 x 8192 |
| 9 |   49152 |        491.52 |        2 x 6 |  5 x 8192 |
+---+---------+---------------+--------------+-----------+

Total Capture duration (choose between 1 and 9): 9

############# Capture duration ##############
#                                           #
# Sample count = 49152                      #
# Capture duration = 491.52 us              #
#                                           #
#############################################
```

The user can then set the pre-trigger by specifying the number of captured data samples before the trigger event:

```
Enter the number of capture samples before trigger activation: 200

###### Capture duration before Trigger ######
#                                           #
# Sample count = 200                        #
# Capture duration = 20.2 us                #
#                                           #
#############################################
```

In the next step, the user can disconnect one GPIO input signal from the DUT circuit and connect it with a fixed value. Please note, that the signal value is set later during the runtime.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Note !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                                   !
! You can override an input or input-vector of your top-level entity using the ILA. !
! Please note that the input will no longer be connected to the FPGA's IO pins.     !
!                                                                                   !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Would you like to implement the input control feature? (y/N): y
--- Inputs DUT "ws2812_gol" ----
+---+--------+--------+---------+
| # |  type  | range  |  Name   |
+---+--------+--------+---------+
| 0 | input  |   1    |   clk   |
| 1 | input  |   1    |  reset  |
| 2 | input  |   1    | reset_2 |
| 3 | input  | [15:0] |  stswi  |
+---+--------+--------+---------+

Select an input signal: 3
```

Then the type of trigger condition (edge or pattern-matching trigger) has to be entered:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Note !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                       !
! There are two default triggers that can be set for exactly one signal: !
!  'rising edge' and 'falling edge'                                     !
! There is also an optional trigger: pattern compare                    !
! With this option, a pattern can be set across the entire bit width,   !
!  determining for each bit whether it should be '1', '0', or 'dc'      !
!  (don't care) to activate the trigger.                                !
! If this function is activated, more hardware is required for the ILA  !
!  and the maximum possible sampling frequency may be reduced.          !
!                                                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Would you like to implement the function for comparing bit patterns? (y/N): N
```

Finally, all entries have been made successfully and a list of the signals to be analyzed is displayed.

```
############ Signals under test #############
#                                           #
# gol_init                                  #
# gol_next_gen                              #
#  [88:81] life_out                         #
#  [78:71] life_out                         #
#  [68:61] life_out                         #
#  [58:51] life_out                         #
#  [48:41] life_out                         #
#  [38:31] life_out                         #
#  [28:21] life_out                         #
#  [18:11] life_out                         #
# [1:0] rgb_color                           #
# [7:0] shift_life_row                      #
# [7:0] ws2812_ram_addr_wr                  #
# [7:0] ws2812_rgb_byte_reg                 #
# [7:0] shift_rgb_byte                      #
# [7:0] ws2812_ram_addr_rd                  #
# data_out                                  #
# run                                       #
#                                           #
#############################################
```

The synthesis is now started automatically with the ILA configuration.

The FPGA bitstream file is generated and is loaded into the FPGA or the flash memory, depending on the `config.py` setup.

```
Execute Synthesis...
Output permanently saved to: C:\Users\df\Desktop\ILA_03_10_2024_clean\gatemate_ila\
    ↪ log\yosys.log

Execute Implementation...
Output permanently saved to: C:\Users\df\Desktop\ILA_03_10_2024_clean\gatemate_ila\
    ↪ log\impl.log

#################### Configuration File ####################
#                                                          #
# save_config\ila_config_ws2812_gol_24-10-14_10-32-20.json #
#                                                          #
############################################################

Upload to FPGA Board...
```

User settings are displayed now before the DUT is put into operation:

```
############ CONFIGURATION NOTE #############
#                                           #
# Trigger at sample no.: 200                #
# Defined analysis frequency: 10000000 Hz   #
#                                           #
#############################################

─────────── All Signals ───────────
+─────+───────────────────────────────+
|   # |                          Name |
+─────+───────────────────────────────+
|   0 |          \ram_to_bit.aserial.run |
|   1 |     \ram_to_bit.aserial.data_out |
|   2 | \ram_to_bit.ws2812_ram_addr_rd[0] |
|   3 | \ram_to_bit.ws2812_ram_addr_rd[1] |
|   4 | \ram_to_bit.ws2812_ram_addr_rd[2] |
|   5 | \ram_to_bit.ws2812_ram_addr_rd[3] |
|   6 | \ram_to_bit.ws2812_ram_addr_rd[4] |
|   7 | \ram_to_bit.ws2812_ram_addr_rd[5] |
|   8 | \ram_to_bit.ws2812_ram_addr_rd[6] |
|   9 | \ram_to_bit.ws2812_ram_addr_rd[7] |
|  10 |     \ram_to_bit.shift_rgb_byte[0] |
|  11 |     \ram_to_bit.shift_rgb_byte[1] |
|  12 |     \ram_to_bit.shift_rgb_byte[2] |
|  13 |     \ram_to_bit.shift_rgb_byte[3] |
|  14 |     \ram_to_bit.shift_rgb_byte[4] |
|  15 |     \ram_to_bit.shift_rgb_byte[5] |
|  16 |     \ram_to_bit.shift_rgb_byte[6] |
|  17 |     \ram_to_bit.shift_rgb_byte[7] |
|  18 |    \golx64.ws2812_rgb_byte_reg[0] |
|  19 |    \golx64.ws2812_rgb_byte_reg[1] |
|  20 |    \golx64.ws2812_rgb_byte_reg[2] |
...
| 104 |         \golx64.life_out_88_81[4] |
| 105 |         \golx64.life_out_88_81[5] |
| 106 |         \golx64.life_out_88_81[6] |
| 107 |         \golx64.life_out_88_81[7] |
| 108 |              \golx64.gol_next_gen |
| 109 |                  \golx64.gol_init |
+─────+───────────────────────────────+

######## current ILA runtime configuration ########
#                                                 #
# Number of sequences: 1                          #
#                                                 #
#  Sequences Number: 1                            #
#     trigger activation: falling edge            #
#     trigger signal:     \ram_to_bit.aserial.run #
#                                                 #
###################################################
```

(For a better overview, the list is abbreviated here.)

The ILA design is now fully configured. ILAcop.py generates the bitstream file for the ILA design and the DUT and uploads it into the GateMate FPGA. Then the DUT is put into operation and ILAcop.py displays the menu that allows the user to control the analysis.

ILAcop.py will control the process by downloading the data and displaying it in the specified wave viewer once the data capturing has been completed.

Depending on how the trigger settings were selected, further capturing can be carried out with subsequent download and display of the data.

```
0 —— exit
1 —— change Trigger
2 —— start capture
3 —— reset ILA (resets the config of the ILA)
4 —— reset DUT (hold the DUT in reset until the capture starts)

Enter your choice: 1
```

8.4 Configuration from a JSON file

Instead of performing the configuration in the interactive shell each time ILAcop.py is started, the complete ILA configuration can be read from a JSON file.

To support this, every time ILA is configured with the interactive shell, the configuration is automatically saved in the `app/save_config/` directory in a JSON file.

The created filename contains the top-level entity name of the DUT plus the date and time of creation in the format

`ila_config_⟨name⟩_⟨year⟩_⟨month⟩_⟨day⟩_⟨hour⟩_⟨minute⟩_⟨second⟩.json`.

This file can of course be renamed and edited if other settings are required than those entered in the interactive configuration run.

For each entry in the JSON file, there is a comment that indicates its function.

The following settings can be changed in the JSON file:

- Selection of signals to be analyzed
- Enable or disable the pattern-matching trigger function
- Number of register levels via which the signals to be analyzed are synchronized
- Frequency and phase shift of the additional PLL (if used)

With the following command, ILA can be started again, using the specified JSON file:

```
$ > python3 ILAcop.py reconfig -l <filename>.json
```

It is also possible to create a JSON file from a DUT, so that the configurations can be made directly in the file, without the need for the interactive shell.

For this, the program must be started with the `-create_json` option, e.g.

```
$ > python3 ILAcop.py config -v C:\Users\df\work\AES_encrypt\src -t
    ↪ aes_spi_top -create_json
```

The interactive shell then only asks for a few settings and otherwise writes default values to the JSON file.

Afterwards, the configuration values must be set inside the JSON file.

The analysis signals are defined as follows:

```
  {
    "Signal_type": "reg",
    "Signal_range": "[127:0]",
    "Signal_name": "data_in",
    "Signal_moduls": "\\encrypt.",
    "selected": [
    "A"
    ]
  },
  {
    "Signal_type": "wire",
    "Signal_range": "[127:0]",
    "Signal_name": "key",
    "Signal_moduls": "\\encrypt.",
    "selected": [
    "127:100"
    ]
  },
  {
    "Signal_type": "wire",
    "Signal_range": "[127:0]",
    "Signal_name": "sub_key",
    "Signal_moduls": "\\encrypt.",
    "selected": [
    "1"
    ]
  },
  {
    "Signal_type": "wire",
    "Signal_range": "[127:0]",
    "Signal_name": "data_out",
    "Signal_moduls": "\\encrypt.",
    "selected": [
    "127:100",
    "88",
    "45:40",
    "10",
    "1"
    ]
  },
```

With `"selected": ['A']` the entire vector is selected. Individual signals can also be specified, as well as subsections of the vector, or a combination of individual signals and subsections within the vector's range.

Afterwards, the desired entries must then be entered in the created file and ILA can be restarted with

```
$ > \python3 ILAcop.py reconfig -l <filename>.json
```

8.5 Restart of ILA on the FPGA at runtime

The communication with the ILA design can be restarted at any time after it has been terminated with the following command:

```
$ > sudo python3 ILAcop.py start
```

This command assumes that the gateware is still loaded in the FPGA. ILAcop.py loads the last setting from the file `gatemate_ila/app/last_upload.txt` and restarts the DUT and the ILA design in the FPGA.

Acronyms

DUT	device under test	5, 7, 9, 10, 13–19, 21, 23–25, 29, 33, 34, 36, 38, 39, 47, 50–52, 54
FPGA	field-programmable gate array	7, 9, 16, 23, 27, 31, 33, 34, 36, 37, 54
GoL	Game of Life	37
GPIO	general purpose input / output	16, 27, 29, 33, 47
ILA	integrated logic analyzer	5–13, 15–21, 23–29, 31, 33, 34, 36–38, 46, 47, 49, 51–54
JSON	JavaScript Object Notation (data interchange format)	10, 15, 23–25, 34, 46, 52, 53
JTAG	Joint Test Action Group	27, 28, 31, 33
PLL	phase-locked loop	16, 24, 38, 52
RAM	random-access memory	14, 16, 17, 24, 29, 30, 37, 38, 47
SPI	Serial Peripheral Interface	5, 8, 27–29, 31, 33
USB	Universal Serial Bus	8, 21, 31, 32
VHDL	Very High Speed Integrated Circuit Hardware Description Language	15, 24, 38
Yosys	Yosys Open SYnthesis Suite	7, 9, 12, 21, 24, 26, 34, 38

GateMate^{TM} FPGA User Guide
Integrated Logic Analyzer
UG1005
August 2025

Cologne
Chip